TRABAJO PRACTICO 2  
Introducción a los Sistemas Operativos

**1. Responda en forma sintética sobre los siguientes conceptos:**

**a. ¿Qué es un Sistema Operativo?**

Un Sistema operativo es un software que actúa como intermediario entre el usuario de la computadora y su hardware.

**b. Enumere qué componentes/aspectos del Hardware son necesarios para cumplir los objetivos de un Sistema Operativo**

Un **Sistema Operativo (SO)** necesita de ciertos componentes de **hardware** para cumplir sus objetivos (administrar recursos, facilitar la interacción con el usuario y ejecutar programas). Los principales son:

1. **Unidad Central de Procesamiento (CPU)**
   * Ejecuta las instrucciones de los programas.
   * El SO debe administrarla (planificación de procesos, multitarea).
2. **Memoria Principal (RAM)**
   * Espacio donde se cargan y ejecutan los programas y datos.
   * El SO gestiona la asignación, liberación y protección de memoria.
3. **Dispositivos de Entrada/Salida (E/S)**
   * Teclado, mouse, monitor, impresoras, pantallas táctiles, etc.
   * El SO se encarga de controlarlos mediante *drivers*.
4. **Dispositivos de Almacenamiento Secundario**
   * Discos duros, SSD, memorias externas.
   * El SO organiza los datos mediante sistemas de archivos.
5. **Dispositivos de Comunicación**
   * Placas de red, Wi-Fi, Bluetooth, puertos USB.
   * Permiten la conectividad y el intercambio de información.
6. **Reloj del sistema (Timer/Clock de hardware)**
   * Permite medir el tiempo, organizar la ejecución de procesos y sincronizar tareas.
7. **Dispositivos de Interrupciones y Controladores**
   * Mecanismo mediante el cual los periféricos avisan al SO que requieren atención (ej: llegada de datos por teclado o red).

**c. Enumere componentes de un Sistema Operativo**

Los componentes de SO son:

* Kernel: Es una porción de código dentro de memoria principal, se encarga de administrar los recursos, manejar memoria, CPU, administrar procesos, comunicación y concurrencia y Gestion de E/S
* Shell: GUI/CUI o CLI
* Herramientas: Editores, compiladores, Liberias, etc

**d. ¿Que es una llamada al sistema (system call)? ¿Cómo es posible implementarlas?**

Es la forma en que los programas de usuario acceden a los servicios del SO. Los parámetros de las llamadas se pueden pasar de varias maneras: Por registro, por bloque o tablas de memoria o por pila, se ejecutan en modo Kernel.

Categorias de SC:

* Control de procesos
* Manejo de archivos
* Manejo de dispositivos
* Mantenimiento de información del sistema
* Comunicaciones.

**e. Defina y diferencia Progama y procesos.**

**Programa**: conjunto de instrucciones o código escrito en un lenguaje de programación que especifica una serie de acciones a realizar por una computadora para llevar a cabo una tarea específica.

* Es estático.
* No tiene program counter.
* Existe desde que se edita hasta que se borra.

**Proceso**: instancia en ejecución de un programa en un sistema operativo.

* Es dinámico.
* Tiene program counter.
* Su ciclo de vida comprende desde que se lo ejecuta hasta que termina.
* Poseen estados y en todo momento están en uno de ellos.

**g. ¿Qué objetivos persiguen los algoritmos de planiﬁcación (scheduling).**

Los **algoritmos de planificación (scheduling)** del sistema operativo tienen como objetivo decidir **qué proceso se ejecuta y cuándo**, buscando un uso eficiente del procesador y de los recursos del sistema.

Los principales objetivos que persiguen son:

1. **Maximizar el uso del procesador (CPU utilization):** evitar que el CPU quede inactivo.
2. **Maximizar el rendimiento (throughput):** lograr que se completen la mayor cantidad de procesos por unidad de tiempo.
3. **Minimizar el tiempo de espera (waiting time):** reducir el tiempo que un proceso permanece en la cola de listos antes de ejecutarse.
4. **Minimizar el tiempo de retorno (turnaround time):** disminuir el tiempo total que transcurre desde que un proceso entra al sistema hasta que finaliza.
5. **Minimizar el tiempo de respuesta (response time):** especialmente importante en sistemas interactivos, reducir el tiempo entre la solicitud de un usuario y la primera respuesta del sistema.
6. **Garantizar equidad (fairness):** que todos los procesos reciban una parte justa de CPU y recursos, evitando inanición (starvation).
7. **Cumplir con prioridades o plazos temporales (deadlines):** en sistemas de tiempo real, asegurar que las tareas críticas se ejecuten en el momento adecuado.

👉 En resumen: buscan **eficiencia, equidad y rapidez en la atención de procesos**, adaptándose según el tipo de sistema (batch, interactivo o tiempo real).

**h. Qué signiﬁca que un algoritmo de scheduling sea apropiativo o no apropiativo (Preemptive o Non-Preemptive)?**

Un **algoritmo de scheduling** puede ser:

**🔹 No apropiativo (Non-Preemptive)**

* Una vez que un proceso obtiene la CPU, **no se le quita hasta que termine o entre en estado de espera** (por E/S, por ejemplo).
* El procesador no interrumpe a un proceso en ejecución para darle lugar a otro.
* Ejemplo: **FIFO (First Come First Served)** o **SJF no apropiativo**.

✅ Ventajas: sencillo, menos sobrecarga de cambios de contexto.

❌ Desventajas: puede generar problemas de **inanición** o tiempos de espera largos si un proceso muy grande se ejecuta antes que otros cortos.

**🔹 Apropiativo (Preemptive)**

* El sistema operativo **puede interrumpir un proceso en ejecución** para ceder la CPU a otro proceso que cumpla con ciertos criterios (por ejemplo, mayor prioridad, menor tiempo estimado de ejecución, o turno en Round Robin).
* Ejemplo: **Round Robin**, **SJF apropiativo (SRTF)**, **Planificación por prioridades apropiativa**.

✅ Ventajas: mayor **equidad** y **tiempos de respuesta más cortos**, ideal en sistemas interactivos.

❌ Desventajas: más **sobrecarga** por cambios de contexto, mayor complejidad de implementación.

👉 En resumen:

* **No apropiativo** → el proceso sigue hasta que termine o espere voluntariamente.
* **Apropiativo** → el SO puede **forzar una interrupción** y reasignar la CPU.

**i. ¿Qué tareas realizan los siguientes módulos de planiﬁcación?:**

**i. Short Term Scheduler:** Determina cual de todos los procesos que están listos para ejecutarse se va a ejecutar a continuación en un ambiente multiprogramado.

**ii. Long Term Scheduler:** Controla el grado de multiprogramación, la cantidad de procesos en memoria. Puede no existir y absorber todo el trabajo el short term

**iii. Medium Term Scheduler:** Saca temporalmente de memoria los procesos que sea necesario para mantener el equilibrio del sistema. Si es necesario reduce el grado de multiprogramación.

**j. ¿Qué tareas realiza el Dispatcher? ¿Y el Loader?**

El distpacher hace cambio de contexto, cambio de modo de ejecución, despacha el proceso elegido por el shor term.

El loader carga en memoria el proceso elegido por el long term.

**k. ¿Qué signiﬁca que un proceso sea “CPU Bound” y “I/O Bound”?**

**CPU Bound:** Son procesos que **usan intensivamente el procesador**. Pasan la mayor parte del tiempo ejecutando **cálculos o instrucciones en CPU**, con pocas operaciones de entrada/salida.

Ejemplos:

* Compilación de un programa grande.
* Procesamiento de imágenes o videos.
* Simulaciones matemáticas.

🔧 Característica: tienden a **saturar el CPU** antes que los dispositivos de E/S.

**I/O Bound:** Son procesos que **dependen mucho de operaciones de entrada/salida (disco, red, teclado, etc.)**.Pasan más tiempo esperando que se complete una operación de E/S que ejecutando en CPU.

Ejemplos:

* Reproducción de música (espera de lectura de datos).
* Consultas a bases de datos.
* Descarga de archivos de Internet.

🔧 Característica: tienden a **bloquearse a menudo** a la espera de la E/S, dejando libre la CPU.

**l. Cuáles son los estados posibles por los que puede atravesar un proceso? ¿Qué representa que un proceso se encuentre en los estados enumerados? Utilizando un diagrama explique las transiciones entre los estados.**

Un proceso, durante su ciclo de vida, puede estar en distintos **estados**:

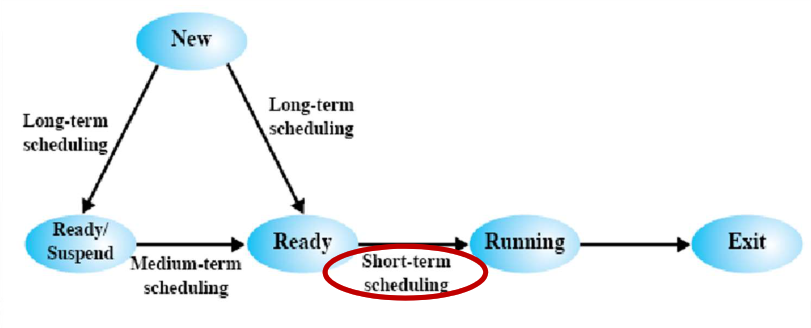
**Nuevo (New):** El proceso ha sido creado, pero aún no está listo para ejecutarse (ej. cargándose en memoria).

**Listo (Ready):** El proceso está cargado en memoria, esperando turno para usar la CPU. Puede ejecutarse, pero el planificador todavía no le asignó el procesador.

**Ejecución (Running):** El proceso está **usando el CPU en ese momento**. Solo un proceso por CPU puede estar en este estado.

**Bloqueado / Espera (Waiting / Blocked):** El proceso está detenido esperando que termine una **operación de E/S** o algún evento externo (ej. lectura de disco, llegada de datos de red).

**Terminado (Terminated / Exit):** El proceso ya completó su ejecución o fue finalizado por el sistema operativo. Sus recursos serán liberados.



**n. Defina Tiempo de retorno (TR) y Tiempo de espera (TE) para un proceso.**

**Tiempo de retorno (TR):** tiempo que transcurre entre que el proceso llega al sistema hasta que completa su ejecución.

**Tiempo de espera (TE):** tiempo que el proceso se encuentra en el sistema esperando, es decir, el tiempo que pasa sin ejecutarse (TR – Tiempo de CPU “Tcpu”)

**o. Defina Tiempo Promedio de Retorno (TPR) y Tiempo Promedio de Espera (TPE) para un lote de JOBS.**

**Tiempo Promedio de Retorno (TPR):** es el promedio de los tiempos de retorno de todos los procesos del lote (suma de TR de cada proceso del lote / cantidad de procesos del lote).

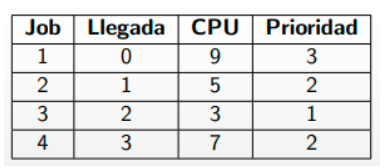
**Tiempo Promedio de Espera (TPE):** Es el promedio de los tiempos de espera de todos los procesos del lote (suma de TE de cada proceso del lote / cantidad de procesos del lote)

**p. Defina tiempo de respuesta.**

El tiempo de respuesta es el tiempo desde la llegada del proceso hasta el momento que usa la cpu por primera vez

2 - Para los siguientes algoritmos de scheduling:

* FCFS (Fisrt Coome First Served)
* SJF (Shortest Job First)
* Round Robin
* Prioridades



FCFS (Fisrt Coome First Served)

* Cuando hay que elegir un proceso para ejecutar, se selecciona el más viejo.
* No favorece a ningún tipo de procesos.
* Los CPU Bound terminan al comenzar su primera ráfaga mientras que los I/O Bound no.
* Es no apropiativo.
* Orden de ejecución: Job 1, Job 2, Job 3, Job 4

SJF (Shortest Job First)

* Selecciona al proceso con la ráfaga más corta.
* Los procesos cortos y los I/o Bound se ven beneficiados.
* Los procesos largos pueden sufrir starvation (inanición) ya que pueden pasar mucho tiempo sin ser seleccionados.
* Es no apropiativo.
* Orden de ejecución: Job 1, Job 3, Job 2, Job 4

Round Robin:

* Política basada en el uso de un reloj (Quantum).
* Cuando un proceso es expulsado de la CPU es colocado al final de la cola de listos y se selecciona otro generando un FIFO circular.
* Existe un “contador” que indica las unidades de CPU en las que el proceso se ejecutó. Cuando este llega a 0 el proceso es expulsado.
* El “contador” puede ser Global o Local (PCB).
* Se puede tener 2 variantes con respecto al valor inicial del “contador” cuando un proceso es asignado a la CPU: Timer Variable o Timer Fijo.
* Es apropiativo.

Prioridades:

* Cada proceso tiene un valor que representa su prioridad, mientras menor sea su valor, mayor prioridad.
* Se selecciona al proceso con mayor prioridad de la cola de listos.
* Puede ser apropiativo o no.
* Existe una cola de listos para cada nivel de prioridad.
* Los procesos de baja prioridad pueden sufrir starvation (inanición).
* Orden de ejecución No Apropiativo: Job 1, Job 3, Job 2, Job 4

**(b) ¿Alguno de ellos requiere algún parámetro para su funcionamiento?**

El algoritmo de Round Robin requiere la definición del tamaño del Quantum.

**(c) Cual es el más adecuado según los tipos de procesos y/o SO**.

- FIFO/FCFS: No favorece a ningún tipo de procesos.

- SJF: Favorece a los I/O Bound y a los procesos cortos.

- Round Robin: Favorece más a los I/O Bound.

- Prioridades: Favorece a los de mayor prioridad.

**(d) Cite ventajas y desventajas de su uso.**

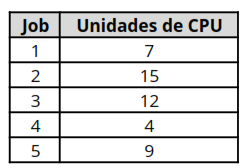
**FIFO** es sencillo de implementar y asegura que los procesos se ejecuten en el orden en que llegan, lo cual es justo para los primeros en la cola. Sin embargo, su principal desventaja es el problema del convoy, donde procesos cortos pueden quedar esperando mucho tiempo si están detrás de procesos largos.

**SJF** minimiza el tiempo promedio de espera al priorizar los procesos más cortos, lo que lo hace eficiente en ciertas cargas de trabajo. No obstante, es difícil predecir el tiempo de ejecución de un proceso, y este enfoque puede ser injusto al favorecer procesos cortos mientras discrimina los largos.

**Round Robin** es equitativo, ya que distribuye el tiempo de CPU de manera uniforme entre los procesos. Es ideal para sistemas interactivos o de tiempo compartido. Sin embargo, tiene como desventaja el aumento del overhead debido al cambio de contexto, y su eficiencia depende de un ajuste adecuado del quantum.

La planificación por **prioridades** permite dar mayor relevancia a los procesos más críticos, lo que la hace flexible y útil en diversos escenarios. Por otro lado, puede generar inanición, donde procesos de baja prioridad no se ejecutan, y su implementación es más compleja en comparación con otros algoritmos.

**3. Dado el siguiente lote de procesos en el que todos arriban al sistema en el instante 0 (cero):**



(**a) Realice los diagramas de Gantt según los siguientes algoritmos de scheduling:**

**i. FCFS (First Come, First Served)**

**ii. SJF (Shortest Job First)   
iii. Round Robin con quantum = 4 y Timer Fijo   
iv. Round Robin con quantum = 4 y Timer Variable**

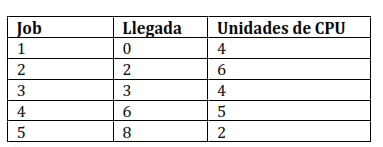
**(b) Para cada algoritmo calcule el TR y TE para cada job así como el TPR y el TPE.**

Solución en la carpeta de Diagramas de Gantt.

**c. En base a los tiempos calculados compare los diferentes algoritmos.**

Para este lote de procesos, el algoritmo con mejores promedios fue el SJF y el que tuvo peores promedios fue el Round Robin de Timer Variable con Quantum 4.

**4. Dado el siguiente lote procesos:**

****

**a. Realice los diagramas de Gantt según los siguientes algoritmos de scheduling:**

1. **FCFS (First Come, First Served)**
2. **SJF (Shortest Job First)**
3. **Round Robin con quantum = 1**
4. **Round Robin con quantum = 6**

**b. Para cada algoritmo calcule el TR y TE para cada job así como el TPR y el TPE.**

Se ejecuto en el archivo diagramas de Gantt.

**c) En base a los tiempos calculados compare los diferentes algoritmos.**

El algoritmo con mejores promedios fue el SJF, mientras que el algoritmo con peor promedio fue el Round Robin de Timer Variable con Quantum 1.

**(d) En el algoritmo Round Robin, que conclusión se puede sacar con respecto al valor del quantum.**

Respecto al valor del Quantum podemos decir que mientras más chico sea, más cambios de contexto habrá, haciendo que los promedios se disparen, y cuanto más grande sea, se comporta de manera similar al FCFS disminuyendo los promedios.

**(e) ¿Para el algoritmo Round Robin, en qué casos utilizaría un valor de quantum alto y que ventaja y desventajas obtendría?**

Se podría utilizar un quantum alto para manejar lotes de procesos CPU Bound ya que estos en lo general ocupan por un largo tiempo los recursos de la CPU, disminuyendo las tasas de tiempo de retorno de todos estos procesos.

La desventaja de esto sería que, si en el lote de procesos nos encontramos procesos cortos o por ejemplo procesos I/O Bound que por lo general no hacen mucho consumo de la CPU, con un quantum alto podríamos generar que las tasas de tiempo de espera se disparen, haciendo que los procesos cortos tengan que esperar muchos ciclos de CPU para poder ejecutarse

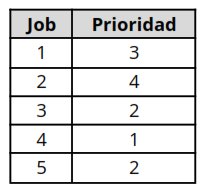
**5. Una variante al algoritmo SJF es el algoritmo SJF apropiativo o SRTF (Shortest Remaining Time First):**

**a. Realice el diagrama de Gantt para este algoritmo según el lote de trabajos del ejercicio 4.**

**b. ¿Nota alguna ventaja frente a otros algoritmos?**

Genera un buen promedio de tiempo de espera, por lo tanto, favorece ventajosamente a procesos I/O Bound ya que estos dependen del tiempo de espera, también los procesos cortos se verían beneficiados.

**6. Suponga que se agregan las siguientes prioridades al lote de procesos del ejercicio 5, donde un menor número indica mayor prioridad:**

****

**a. Realice el diagrama de Gantt correspondiente al algoritmo de planificación por prioridades según las variantes:**

1. **No Apropiativa**
2. **Apropiativa**

**b. Calcule el TR y TE para cada job así como el TPR y el TPE.**

**c. ¿Nota alguna ventaja frente a otros algoritmos? ¿Bajo qué circunstancias lo utilizaría y ante qué situaciones considera que la implementación de prioridades podría no ser de mayor relevancia?**

Las ventajas es que para tareas críticas del sistema operativo que necesiten ser ejecutadas inmediatamente, van a ser muy buenos, además de que brindan una mayor adaptabilidad en nuestro sistema si decidimos llevar una organización por prioridades. Los casos donde los utilizaría son justamente los mencionados en las ventajas.

**7. Inanición (Starvation)**

**a. ¿Qué signiﬁca?**

Significa que el proceso esta en una situación donde no puede avanzar o ejecutarse debido a la competencia por recursos del sistema entre los diferentes procesos. Ocurro cuando un proceso no puede obtener los recursos necesarios para su ejecución porque otros los están acaparando.

(**b) ¿Cuál/es de los algoritmos vistos puede provocarla?**

De los algoritmos vistos los que pueden generar inanición serían el SJF, el STRF y el de Prioridades. Los 2 primeros generándola para procesos largos y el tercero en procesos de baja prioridad.

**(c) ¿Existe alguna técnica que evite la inanición para el/los algoritmos mencionados en b?**

* **Envejecimiento (Aging):** consiste en aumentar la prioridad de un proceso a medida que pasa el tiempo sin ser seleccionado para su ejecución.
* **Establecimiento de límites de espera**: consiste en establecer límites de tiempo máximo para la espera de un proceso en la cola de listos, haciendo que, si llega a este límite, pase a ejecutarse.
* **Prioridades dinámicas:** podemos tener sistemas que ajusten las prioridades de los procesos dinámicamente según el comportamiento y las necesidades del proceso.

**8. Los procesos, durante su ciclo de vida, pueden realizar operaciones de I/O como lecturas o escrituras a disco, cintas, uso de impresoras, etc.**

**El SO mantiene para cada dispositivo, que se tiene en el equipo, una cola de procesos que espera por la utilización del mismo (al igual que ocurre con la Cola de Listos y la CPU, ya que la CPU es un dispositivo más).**

**Cuando un proceso en ejecución realiza una operación de I/O el mismo es expulsado de la CPU y colocado en la cola correspondiente a el dispositivo involucrado en la operación.**

**El SO dispone también de un “I/O Scheduling” que administrada cada cola de dispositivo a través de algún algoritmo (FCFS, Prioridades, etc.). Si al colocarse un proceso en la cola del dispositivo, la misma se encuentra vacía el mismo será atendido de manera inmediata, caso contrario, deberá esperar a que el SO lo seleccione según el algoritmo de scheduling establecido.**

**Los mecanismos de I/O utilizados hoy en día permiten que la CPU no sea utilizada durante la operación, por lo que el SO puede ejecutar otro proceso que se encuentre en espera una vez que el proceso bloqueado por la I/O se coloca en la cola correspondiente.**

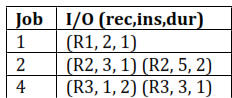
**Cuando el proceso finaliza la operación de I/O el mismo retorna a la cola de listos para competir nuevamente por la utilización de la CPU.**

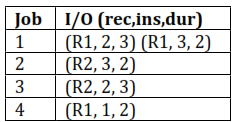
**Para los siguientes algoritmos de Scheduling:**

* **FCFS**
* **Round Robin con quantum = 2 y timer variable.**

**Y suponiendo que la cola de listos de todos los dispositivos se administra mediante FCFS, realice los diagramas de Gantt según las siguientes situaciones:**

**(a) Suponga que al lote de procesos del ejercicio 6 se agregan las siguientes operaciones de entrada salida:**



**(b) Suponga que al lote de procesos del ejercicio 5 se agregan las siguientes operaciones de entrada salida**

A Y B se realizaron en el Excel adjunto.

**9 . Algunos algoritmos pueden presentar ciertas desventajas cuando en el sistema se cuenta con procesos ligados a CPU y procesos ligados a entrada salida. Analice las mismas para los siguientes algoritmos:**

**a. Round Robin**

**Ventaja general:** RR busca dar equidad entre procesos, repartiendo el CPU en quantums iguales.

**Problema con CPU-Bound vs I/O-Bound:** Los procesos I/O-Bound suelen usar poco tiempo de CPU y enseguida hacen I/O → esto los hace más sensibles a la latencia: si deben esperar demasiado en la cola, se retrasa su entrada/salida, afectando el rendimiento global.

Los procesos CPU-Bound ocupan su quantum completo siempre → terminan generando más cambios de contexto, lo que aumenta la sobrecarga del sistema.

**Resultado:** los I/O-Bound pueden terminar “castigados” con esperas innecesarias, aunque en teoría deberían tener prioridad para mantener al sistema productivo (el CPU y el dispositivo I/O trabajando en paralelo).En cuando a los procesos I/O bound este proceso es mas adecuado, pero presenta algunas desventajas como pueden ser problemas de inanición, ineficiendia en la utilización de la CPU y retrasos en la finalización de estos procesos ya que pueden ser interrumpidos por el context switch.

**b. SRTF (Shortest Remaining Time First)**

**Ventaja general**: en teoría favorece a los procesos cortos y mejora el tiempo promedio de retorno.

**Problema con CPU-Bound vs I/O-Bound**: Los procesos **I/O-Bound** tienden a ser cortos en CPU, entonces el algoritmo **los beneficia**, dándoles prioridad sobre los CPU-Bound.

Pero los procesos **CPU-Bound** largos pueden sufrir **espera excesiva** o incluso **inanición** (starvation) si continuamente llegan procesos cortos (I/O-Bound o jobs breves).

En sistemas con mezcla, el CPU-Bound puede avanzar muy lentamente, generando falta de equidad.

✅ **Resumen comparativo**:

* **RR**: equitativo, pero introduce latencia a los I/O-Bound y sobrecarga de cambios de contexto.
* **SRTF**: favorece a los I/O-Bound, pero puede perjudicar seriamente a los CPU-Bound, que corren riesgo de inanición.

**10. Para equiparar la desventaja planteada en el ejercicio 11), se plantea la siguiente modificación al algoritmo:**

**Algoritmo VRR (Virtual Round Robin): Este algoritmo funciona igual que el Round Robin, con la diferencia que cuando un proceso regresa de una I/O se coloca en una cola auxiliar. Cuando se tiene que tomar el próximo proceso a ejecutar, los procesos que se encuentra en la cola auxiliar tienen prioridad sobre los otros. Cuando se elige un proceso de la cola auxiliar se le otorga el procesador por tantas unidades de tiempo como le falto ejecutar en su ráfaga de CPU anterior, esto es, se le otorga la CPU por un tiempo que surge entre la diferencia del quantum original y el tiempo usado en la última ráfaga de CPU.**

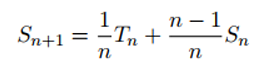
**(a) Analice el funcionamiento de este algoritmo mediante un ejemplo. Marque en cada instante en que cola se encuentran los procesos.**

**(b) Realice el ejercicio 9)a) nuevamente considerando este algoritmo, con un quantum de 2 unidades y Timer Variable.**

Solución en la carpeta de Diagramas de Gantt.

**11. Suponga que el kernel utiliza un algoritmo de VRR con Timer Variable para el planificar sus procesos. Para ello, el quantum es representado por un contador, que es decrementado en 1 unidad cada vez que ocurre una interrupción de reloj. ¿Bajo este esquema, puede suceder que el quantum de un proceso nunca llegue a 0 (cero)? Justifique su respuesta.**

Sí, esto podría ocurrir si un proceso termina su ejecución en una ráfaga de CPU antes de que se termine su Quantum. Sin embargo, en condiciones normales y con un sistema operativo bien diseñado, esto no ocurrirá.

**12. El algoritmo SJF (y SRTF) tiene como problema su implementación, dada la dificultad de conocer la duración de la próxima ráfaga de CPU. Es posible realizar una estimación de la próxima, utilizando la media de las ráfagas de CPU para cada proceso. Así, por ejemplo, podemos tener la siguiente formula:**

**Donde:**

**Ti = duración de la ráfaga de CPU i-ésima del proceso.   
Si = valor estimado para el i-ésimo caso   
Si = valor estimado para la primera ráfaga de CPU. No es calculado.**

**(a) Suponga un proceso cuyas ráfagas de CPU reales tienen como duración: 6, 4, 6, 4, 13, 13, 13. Calcule que valores se obtendrían como estimación para las ráfagas de CPU del proceso si se utiliza la fórmula 1, con un valor inicial estimado de S1=10.**

**La fórmula anterior 1 les da el mismo peso a todos los casos (siempre calcula la media). Es posible reescribir la formula permitiendo darle un peso mayor a los casos más recientes y menor a casos viejos (o viceversa). Se plantea la siguiente formula:**

**Con 0 < α <1.**

Se tienen las ráfagas reales del proceso:  
T = {6, 4, 6, 4, 13, 13, 13}  
y un valor inicial estimado:  
S1 = 10

**Fórmula 1 – Media de todas las ráfagas previas**

Sn+1 = (1/n) \* Tn + ((n-1)/n) \* Sn

**Cálculo paso a paso:**

* S2 = (1/1)\*6 + (0/1)\*10 = 6
* S3 = (1/2)\*4 + (1/2)\*6 = 2 + 3 = 5
* S4 = (1/3)\*6 + (2/3)\*5 = 2 + 3.33 = 5.33
* S5 = (1/4)\*4 + (3/4)\*5.33 = 1 + 4 = 5
* S6 = (1/5)\*13 + (4/5)\*5 = 2.6 + 4 = 6.6
* S7 = (1/6)\*13 + (5/6)\*6.6 = 2.17 + 5.5 = 7.67
* S8 = (1/7)\*13 + (6/7)\*7.67 = 1.86 + 6.57 = 8.43

**Valores obtenidos:**  
S = {10, 6, 5, 5.33, 5, 6.6, 7.67, 8.43}

**Fórmula 2 – Promedio exponencial ponderado**

Sn+1 = α\*Tn + (1-α)\*Sn  
(con α = 0.5)

**Cálculo paso a paso:**

* S2 = 0.5*6 + 0.5*10 = 3 + 5 = 8
* S3 = 0.5*4 + 0.5*8 = 2 + 4 = 6
* S4 = 0.5*6 + 0.5*6 = 3 + 3 = 6
* S5 = 0.5*4 + 0.5*6 = 2 + 3 = 5
* S6 = 0.5*13 + 0.5*5 = 6.5 + 2.5 = 9
* S7 = 0.5*13 + 0.5*9 = 6.5 + 4.5 = 11
* S8 = 0.5*13 + 0.5*11 = 6.5 + 5.5 = 12

**Valores obtenidos:**  
S = {10, 8, 6, 6, 5, 9, 11, 12}

**b) ¿Para qué valores de α\alphaα se tienen en cuenta más los casos recientes?**

* Si α es **cercano a 1** (por ejemplo 0.70.70.7–0.90.90.9), la estimación **da mucho peso** a la última ráfaga Tn​: responde **rápido** a cambios recientes.
* Si α es **cercano a 0** (por ejemplo 0.10.10.1–0.30.30.3), la estimación **da poco peso** a lo último y prioriza el histórico: es **más estable** y reacciona lentamente.
* En términos prácticos: valores grandes de α → consideran más los casos recientes; valores pequeños → consideran más el pasado.

**13. Colas Multinivel**   
**Actualmente los algoritmos de planificación vistos se han ido combinando para formar algoritmos más eficientes. Así surge el algoritmo de Colas Multinivel, donde la cola de procesos listos es dividida en varias colas, teniendo cada una su propio algoritmo de planificación.**

**a. Suponga que se tiene dos tipos de procesos: Interactivos y Batch. Cada uno de estos procesos se coloca en una cola según su tipo. ¿Qué algoritmo de los vistos utilizará para administrar cada una de estas colas?**

**A su vez, se utiliza un algoritmo para administrar cada cola que se crea. Así, por ejemplo, el algoritmo podría determinar mediante prioridades sobre qué cola elegir un proceso.**

**b. Para el caso de las dos colas vistas en a: ¿Qué algoritmo utilizaría para planiﬁcarlas?**

**a. Algoritmo para administrar cada cola según el tipo de proceso**

Tenemos dos tipos de procesos:

**Interactivos** (procesos que requieren respuesta rápida, como editores, juegos, terminales, etc.).

* Para estos procesos queremos que **la respuesta sea rápida**, por lo que el **algoritmo Round Robin (RR)** es ideal.
* Round Robin asigna **quantum de CPU a cada proceso por turnos**, asegurando que ninguno espere demasiado.

**Batch** (procesos que no requieren interacción inmediata, como compilaciones o cálculos largos).

* Para estos procesos buscamos **eficiencia en el uso de CPU**, no necesariamente respuesta rápida.
* Aquí se puede usar **First-Come, First-Served (FCFS)**, ya que los procesos batch pueden esperar y no es crítico que tengan un turnaround mínimo inmediato.

✅ **Resumen a:**

* Interactivos → Round Robin
* Batch → FCFS

**b. Algoritmo para planificar entre colas**

Ahora, si tenemos varias colas (una de interactivos y otra de batch), necesitamos decidir **qué cola tomar primero cuando hay procesos listos en ambas**. Aquí entran las **estrategias de planificación de colas multilevel**:

**1) Prioridad fija de colas:**

* Cada cola tiene una prioridad.
* La CPU siempre toma procesos de la cola de mayor prioridad.
* Ejemplo: Interactivos > Batch (los interactivos son más importantes).

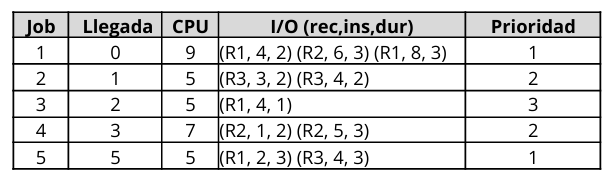
**2) Rotación entre colas (Time-slicing entre colas):**

* Se da un tiempo a cada cola y luego se rota a la siguiente.
* Esto evita que las colas de menor prioridad se queden “hambrientas”.

En un escenario clásico **Interactivos vs Batch**, lo común es usar:

* **Prioridad de colas**: la cola de interactivos tiene **mayor prioridad** que la de batch.
* Dentro de cada cola, se aplica el algoritmo correspondiente (RR para interactivos, FCFS para batch).

**14. Suponga que en un Kernel se utiliza un algoritmo de planificación de colas multinivel. El mismo cuenta con 3 colas de procesos listos, en las que los procesos se encolan en una u otra según su prioridad. Hay 3 prioridades (1 , 2 , 3), donde un menor número indica mayor prioridad. Se utiliza el algoritmo de prioridades para la administración entre las colas. Se tiene el siguiente lote de procesos a ser procesados con sus respectivas operaciones de I/O:**

****

**Suponiendo que las colas de cada recurso (dispositivo de entrada/salida) se administran a través de FCFS y que cada cola de procesos listos se administra por medio de un algoritmo RR con un quantum de 3 unidades, realice un diagrama de Gantt:**

**a. Asumiendo que NO hay apropiación entre los procesos.**

**b. Asumiendo que hay apropiación entre los procesos.**

Realizado en el archivo diagramas de Gantt.

**15. En el esquema de Colas Multinivel, cuando se utiliza un algoritmo de prioridades para administrar las diferentes colas los procesos pueden sufrir starvation. La técnica de envejecimiento se puede aplicar a este esquema, haciendo que un proceso cambie de una cola de menor prioridad a una de mayor prioridad, después de cierto periodo de tiempo que el mismo se encuentra esperando en su cola. Luego de llegar a una cola en la que el proceso llega a ser atendido, el mismo retorna a su cola original.**

**Por ejemplo: Un proceso con prioridad 3 está en cola su cola correspondiente. Luego de X unidades de tiempo, el proceso se mueve a la cola de prioridad 2. Si en esta cola es atendido, retorna a su cola original, en caso contrario luego de sucederse otras X unidades de tiempo el proceso se mueve a la cola de prioridad 1. Esta última acción se repite hasta que el proceso obtiene la CPU, situación que hace que el mismo vuelva a su cola original. Para los casos a y b del ejercicio 15 realice el diagrama de Gantt considerando además que se tiene un envejecimiento de 4 unidades.**

Realizado en el archivo diagramas de Gantt.

**16. La situación planteada en el ejercicio 15, donde un proceso puede cambiar de una cola a otra, se la conoce como Colas Multinivel con Realimentación. Suponga que se quiere implementar un algoritmo de planificación que tenga en cuenta el tiempo de ejecución consumido por el proceso, penalizando a los que más tiempo de ejecución tienen. (Similar a la tarea del algoritmo SJF que tiene en cuenta el tiempo de ejecución que resta). Utilizando los conceptos vistos de Colas Multinivel con Realimentación indique qué colas implementaría, qué algoritmo usaría para cada una de ellas así como para la administración de las colas entre sí. Tenga en cuenta que los procesos no deben sufrir inanición.**

Podria implementar tres colas, cada una con un nivel de prioridad. Los procesos mas largos tendría prioridad menor que los procesos mas cortos. Cada cola indivualmente se maneja con un RR de quantum chico para favorecer a los procesos de menor duración y para solucionar el problema de inanición de los procesos largos. Para administrar las colas usaría un algoritmo por prioridades con aging para que los procesos mas largos vayan escalando en prioridad a medida que los procesos mas chicos vayan terminando su ejecución.

**17. A cuáles de los siguientes tipos de trabajos:**

* **cortos acotados por CPU**
* **cortos acotados por E/S**
* **largos acotados por CPU**
* **largos acotados por E/S**

**benefician las siguientes estrategias de administración:**

**a. prioridad determinada estáticamente con el método del más corto primero (SJF)**

**b. prioridad dinámica inversamente proporcional al tiempo transcurrido desde la última operación de E/S.**

**a. Prioridad determinada estáticamente con el método del más corto primero (SJF)**

**Qué es SJF:** Se priorizan los trabajos de menor duración estimada en CPU.

**Beneficiados:**

* **Trabajos cortos acotados por CPU** → son los más favorecidos, ya que el planificador siempre selecciona primero los más breves en CPU.
* **Trabajos cortos acotados por E/S** → también se benefician indirectamente, porque al ser cortos no esperan demasiado para entrar a ejecutar.

**Perjudicados:**

* **Trabajos largos (CPU o E/S)** → quedan relegados porque siempre hay trabajos cortos que “se les adelantan” (problema de inanición o starvation).

**b. Prioridad dinámica inversamente proporcional al tiempo transcurrido desde la última operación de E/S**

**Qué significa:** Cuanto más tiempo lleva un trabajo sin hacer E/S, más baja su prioridad. Es decir, se favorecen los que interactúan con E/S con frecuencia.

**Beneficiados:**

* **Trabajos cortos acotados por E/S** → hacen E/S seguido, por lo que su prioridad siempre se renueva y entran rápido a CPU.
* **Trabajos largos acotados por E/S** → también reciben un trato preferencial frente a los de CPU intensivo, porque el sistema busca evitar que procesos E/S se queden esperando.

**Perjudicados:**

* **Trabajos acotados por CPU (cortos o largos)** → como pasan mucho tiempo sin realizar E/S, su prioridad cae y esperan más para ejecutarse.

**18. Analizar y explicar por qué si el quantum en Round-Robin se incrementa sin límite, el algoritmo de planificación se aproxima a FIFO.**

Si el quantum es grande, tanto como para que cada proceso termine o salga de la CPU para antender su I/O, se elimina la rotación que plantea el algoritmo RR y la convierte en una cola circular FIFO.

**19. Dado los siguientes programas en un pseudo-código que simula la utilización de llamadas al sistema para crear procesos en Unix/Linux, indicar qué mensajes se imprimirán en pantalla (sin importar el orden en que saldrían):**

**a. Caso 1**

printf(“hola”)   
x = fork()   
if x < 1 {   
 execv(“ls”)   
 printf(“mundo”)   
 exit(0)   
} exit(0)

**Paso a paso**

**1) printf("hola");**

* Escribe "hola" en el buffer de salida estándar. Como no hay \n, puede quedar en buffer hasta que el proceso termine o haga un flush.

**2) x = fork();**

* Se crea un **nuevo proceso (child)**.
* Ahora tenemos:
  + - **Padre** → x > 0 (el valor es el PID del hijo).
    - **Hijo** → x == 0.

⚠️ Los dos procesos tienen su propia copia del buffer con "hola".

**3) if (x < 1) { ... }**

* + El **child** (porque x == 0) entra en el bloque.
  + El **parent** (porque x > 0) no entra.

**Dentro del if (ejecuta solo el hijo)**

* **execv("ls");**
  + Si tiene éxito → el proceso hijo deja de ejecutar este código y pasa a ser el programa ls.
  + Eso significa que **no vuelve nunca** a este bloque ni imprime "mundo".
  + Además, el buffer de "hola" que tenía el hijo se pierde (porque la memoria del proceso se reemplaza).
* **printf("mundo");**
  + Solo se ejecuta si execv("ls") falla (ruta incorrecta, permisos, etc.).
* **exit(0);**
  + Si execv falló y se llega acá, se hace flush de los buffers del hijo: se imprimiría la copia de "hola" y también "mundo".

**Después del if (ejecuta solo el padre)**

* **exit(0);**
  + El padre sale, y en ese momento se hace flush de su buffer.
  + Resultado: imprime su copia de "hola".

**Conclusión Caso 1**

* **Caso normal (execv funciona bien):**
  + El padre imprime **hola**.
  + El hijo ejecuta ls y muestra el listado de archivos.
  + "mundo" no aparece.
  + 👉 Salida:
    - hola
    - salida de ls
* **Caso alternativo (execv falla en el hijo):**
  + El padre imprime su **hola**.
  + El hijo hace flush de su copia de "hola" y también "mundo".
  + 👉 Salida:
    - hola (del padre)
    - hola (del hijo)
    - mundo (del hijo)

**b. Caso 2**

printf("hola");  
x = fork();  
if (x < 1) {

execv("ps");

printf("mundo");

exit(0);

}  
execv("ls");  
printf("fin");  
exit(0);

**Paso a paso**

1. printf("hola");
   * Se escribe "hola" en el buffer de salida.
   * No hay \n, así que probablemente quede en el buffer hasta que el proceso termine (pero se va a imprimir igual al hacer exit o exec que hace flush).
2. x = fork();
   * Se crean **dos procesos**:
     + **Padre**: x > 0.
     + **Hijo**: x = 0.
3. if (x < 1) { ... }
   * El **hijo** entra al bloque.
   * Llama a execv("ps"):
     + Si funciona → el hijo se convierte en el programa ps (que lista procesos). Nunca ejecuta printf("mundo") ni exit(0).
     + Si falla → imprime "mundo" y hace exit(0).
4. El **padre** (como x > 0) **no entra al if**, sigue con:
   * execv("ls"); → el padre se convierte en ls y muestra el listado de archivos. Nunca llega a printf("fin").

**Conclusión caso 2**

* **Normal (execv funciona bien):**
  + Se imprime "hola" (una sola vez).
  + Se ve la salida de ps (del hijo).
  + Se ve la salida de ls (del padre).
  + **No aparecen ni "mundo" ni "fin".**
* **Si algún exec falla:**
  + Podrían aparecer "mundo" o "fin", y hasta dos "hola" (uno del padre y otro del hijo), como te expliqué en el caso anterior.

**c. Caso 3**

printf("Anda a rendir el Primer Parcial de Promo!");  
newpid = fork();  
if (newpid == 0) {

printf("Estoy comenzando el Examen");

execv("ps");

printf("Termine el Examen");

}  
printf("¿Como te fue?");  
exit(0);  
printf("Ahora anda a descansar");

**Paso a paso**

1. printf("Anda a rendir el Primer Parcial de Promo!");
   * Se guarda en el buffer (sin \n).
2. newpid = fork();
   * Se crean **dos procesos**:
     + **Padre:** newpid > 0.
     + **Hijo:** newpid == 0.
3. **Hijo (newpid == 0)**:
   * printf("Estoy comenzando el Examen");
   * execv("ps"); → si funciona, el hijo se convierte en ps y no ejecuta más líneas.
   * Si fallara, imprimiría "Termine el Examen" y luego terminaría.
4. **Padre (newpid > 0)**:
   * Saltea el if y ejecuta:
     + printf("¿Como te fue?");
     + exit(0); → termina el proceso padre, y **nunca llega** a "Ahora anda a descansar".

**Conclusión caso 3**

* **Normal (execv funciona bien):**
  + "Anda a rendir el Primer Parcial de Promo!"
  + "Estoy comenzando el Examen" (del hijo)
  + La salida del programa ps (del hijo tras el exec)
  + "¿Como te fue?" (del padre)
  + **No aparece "Termine el Examen" ni "Ahora anda a descansar".**
* **Si execv("ps") fallara en el hijo:**
  + Además se vería "Termine el Examen".

**✅ Resumen general**

* **Caso 2 (normal):**
  + hola
  + salida de ps
  + salida de ls
* **Caso 3 (normal):**
  + Anda a rendir el Primer Parcial de Promo!
  + Estoy comenzando el Examen
  + salida de ps
  + ¿Como te fue?

**22. ¿Qué es la MMU y qué funciones cumple?**

La **MMU** (del inglés *Memory Management Unit* o Unidad de Gestión de Memoria) es un componente de hardware del procesador —aunque en algunos sistemas puede implementarse en software— que se encarga de **administrar y controlar el acceso a la memoria principal (RAM)** por parte de la CPU y los procesos.

**Funciones principales de la MMU:**

1. **Traducción de direcciones**
   * Convierte las **direcciones lógicas (virtuales)** generadas por los programas en **direcciones físicas** de la memoria RAM.
   * Esto permite que cada proceso “crea” que tiene su propia memoria continua, aunque físicamente no sea así.
2. **Protección de memoria**
   * Garantiza que un proceso no pueda acceder a la memoria de otro proceso o del propio sistema operativo.
   * Si se intenta acceder a un área prohibida, la MMU genera una excepción (*segmentation fault*).
3. **Paginación y segmentación**
   * Implementa técnicas de administración de memoria como **paginación** y **segmentación**, que facilitan la multitarea y el uso eficiente de la memoria.
   * Permite cargar en RAM solo las partes necesarias de un programa, usando también memoria secundaria (swap).
4. **Soporte para memoria virtual**
   * Da la ilusión de que hay más memoria disponible de la que físicamente existe, usando disco duro como extensión (archivo/página de swap).
5. **Control de cachés y permisos**
   * Maneja bits de control como **R/W (lectura/escritura)**, **E (ejecución)** o **presencia** de las páginas, para definir cómo puede usarse cada bloque de memoria.

**23. ¿Que representa el espacio de direcciones de un proceso?**

El **espacio de direcciones de un proceso** es el **conjunto de direcciones de memoria que ese proceso puede usar** durante su ejecución.

👉 En otras palabras, es la “visión” que el proceso tiene de la memoria, organizada de forma lógica por el sistema operativo y la **MMU**.

**Características principales:**

* **Independencia:** Cada proceso tiene su propio espacio de direcciones, aislado del de los demás, lo que garantiza seguridad y estabilidad.
* **Direcciones virtuales:** El proceso maneja direcciones virtuales, que la MMU traduce a direcciones físicas en la RAM.
* **Estructura típica (en sistemas con memoria virtual):**
  1. **Código (text segment):** Instrucciones del programa.
  2. **Datos estáticos (data segment):** Variables globales y estáticas.
  3. **Heap:** Área de memoria dinámica (malloc, new).
  4. **Stack (pila):** Variables locales y control de llamadas a funciones.
  5. **Bibliotecas compartidas (shared libraries):** como DLL o .so cargadas en memoria.

**24. Explique y relacione los siguientes conceptos: Dirección Lógica o Virtual Dirección Física**

**📌 Dirección lógica o virtual**

* Es la dirección de memoria que **genera la CPU** cuando un proceso ejecuta una instrucción.
* Es la que “ve” el programa, pero **no corresponde directamente a una posición real en la RAM**.
* Permite que cada proceso tenga la ilusión de disponer de toda la memoria para sí mismo.
* Ejemplo: un programa puede acceder a la dirección lógica 0x00400000, sin importar dónde se cargue realmente en la memoria física.

**📌 Dirección física**

* Es la **dirección real en la RAM** donde se almacenan los datos o instrucciones.
* La memoria física es única y compartida entre todos los procesos.
* Solo la **MMU (Unidad de Gestión de Memoria)** y el sistema operativo saben qué dirección lógica corresponde a qué dirección física.

**📌 Relación entre ambas**

* El programa usa **direcciones lógicas/virtuales**.
* La **MMU traduce** esas direcciones a **direcciones físicas** mediante tablas de páginas o segmentos.
* Esta traducción permite:  
  ✅ Multiprogramación (varios procesos usan la misma memoria sin interferir).  
  ✅ Protección de memoria (un proceso no pisa la memoria de otro).  
  ✅ Memoria virtual (usar disco como extensión de la RAM).

**25. En la técnica de Particiones Múltiples, la memoria es dividida en varias particiones y los espacios de direcciones de los procesos son ubicados en estas, siempre que el tamaño del mismo sea menor o igual que el tamaño de la partición. Al trabajar con particiones se pueden considerar 2 métodos (independientes entre sí):   
Particiones Fijas   
Particiones Dinámicas**

**a. Explique cómo trabajan estos 2 métodos. Cite diferencias, ventajas y desventajas.**

**b. ¿Qué información debe disponer el Kernel para poder administrar la memoria principal con estos métodos?**

**c. Realice un gráfico indicando cómo se realiza la transformación de direcciones lógicas a direcciones físicas.**

**a) Métodos de particiones múltiples**

**Particiones Fijas**

* La memoria se divide en **bloques de tamaño fijo** al arrancar el sistema.
* Cada partición puede contener **un solo proceso**.
* Si el proceso es más pequeño que la partición → se desperdicia memoria (**fragmentación interna**).
* Si el proceso es más grande que cualquier partición → **no puede cargarse**.

**Ventajas:**

* Administración simple.
* Bajo costo en tiempo de asignación (rápido de decidir dónde cargar).

**Desventajas:**

* Poca flexibilidad (los tamaños están predefinidos).
* Fragmentación interna.
* Limita el tamaño máximo de proceso al de la partición más grande.

**Particiones Dinámicas**

* La memoria se divide **a demanda**, asignando particiones según el tamaño de cada proceso.
* El sistema crea particiones de diferentes longitudes en tiempo de ejecución.
* Produce **fragmentación externa** (espacios libres pequeños entre particiones).
* Requiere técnicas como **compactación** o **algoritmos de ubicación**: *First Fit, Best Fit, Worst Fit*.

**Ventajas:**

* Mayor flexibilidad → los procesos pueden ocupar exactamente lo que necesitan.
* Aprovecha mejor la memoria.

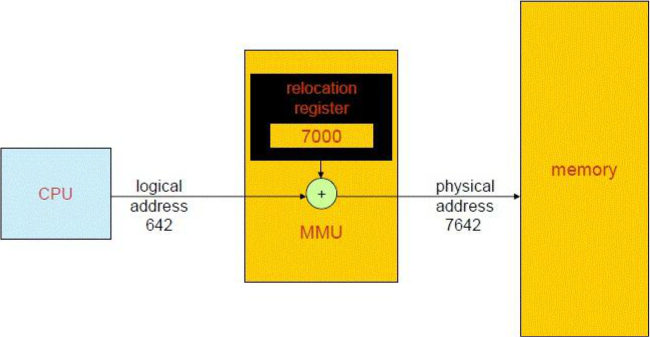
**Desventajas:**

* Fragmentación externa.
* Necesidad de compactar periódicamente (costoso).
* Administración más compleja.

**b) Información que debe manejar el Kernel**

Para administrar memoria en cualquiera de los dos métodos:

* **Tamaño total de la memoria disponible.**
* **Lista de particiones:**
  + Dirección de inicio y fin.
  + Tamaño de cada partición.
  + Estado (*ocupada/libre*).
  + Proceso que la ocupa (si corresponde).
* **Mapa de memoria libre/ocupada** (bitmaps, listas enlazadas, tablas).
* Estrategia de ubicación para asignar memoria (en dinámicas: *First Fit*, *Best Fit*, etc.).



**26. Al trabajar con particiones fijas, los tamaños de las mismas se pueden considerar:**

**➢ Particiones de igual tamaño.**

**➢ Particiones de diferente tamaño.**

**Cite ventajas y desventajas entre las alternativas.**

**📌 Particiones Fijas de Igual Tamaño**

**✅ Ventajas**

* Administración muy simple (todas las particiones son idénticas).
* Fácil de implementar y bajo costo en tiempo de asignación.
* Útil si se espera que la mayoría de los procesos tengan **tamaños similares**.

**❌ Desventajas**

* **Fragmentación interna alta**: si un proceso es más chico que la partición, desperdicia memoria.
* Procesos grandes que superen el tamaño de la partición **no pueden cargarse**, aunque haya suficiente memoria repartida en varias particiones.
* Poco flexible.

**📌 Particiones Fijas de Diferente Tamaño**

**✅ Ventajas**

* Permite **adaptar mejor los procesos** a particiones de distintos tamaños.
* Reduce la fragmentación interna comparado con las de igual tamaño.
* Puede aprovechar mejor la memoria si se diseñan particiones con base en la distribución de tamaños de procesos que se espera ejecutar.

**❌ Desventajas**

* La **asignación es más compleja** (hay que decidir qué proceso entra en qué partición).
* Puede producir **fragmentación interna** si el proceso es mucho más pequeño que la partición elegida.
* Puede generar **inequidad**: algunos procesos siempre terminan en particiones grandes, **dejando menos espacio para otros.**

**27. Fragmentación. Ambos métodos de particiones presentan el problema de la fragmentación:**

**Fragmentación Interna (Para el caso de Particiones Fijas)**

**Fragmentación Externa (Para el caso de Particiones Dinámicas)**

**a. Explique a qué hacen referencia estos 2 problemas**

**b. El problema de la Fragmentación Externa es posible de subsanar. Explique una posible técnica que permita al Kernel mitigar este problema.**

a) La fragmentación interna (particiones fijas) Ocurre cuando un proceso no utiliza toda la memoria asignada en la partición. Es característica de las particiones de tamaño fijo, donde la memoria desperdiciada esta dentro de la partición asignada

La fragmentación Externa (particiones dinámicas) ocurre cuando la memoria libre esta dispersa en pequeños bloques que no son contiguos, por lo que un proceso grande no puede encontrar un bloque lo suficientemente grande. Por ejemplo después de cargar y descargar varios procesos de tamaños distintos, quedan huecos libres pequeños, aunque la suma total de memoria libre sea suficiente. Es típica de particiones de tamaño variable.

b) La técnica es la compactación. El Kernel remueve los procesos cargados en memoria de manera que se unan todos los bloques libres en un solo bloque. Esto permite que procesos grandes puedan cargarse aunque antes no hubiera un bloque contiguo suficiente.

Otra opción es el uso de algoritmos de asignación eficientes como Best Fit, Worst Fit, First Fit y Next Fit que tratan de minimizar la dispersión de bloques libres.

**28. Analice y describa cómo funcionan las siguientes técnicas de asignación de particiones: Best Fit, Worst Fit, First Fit y Next Fit. Tenga en cuenta que información debe mantener el Kernel. Indique, para los métodos de particiones dinámicas y ﬁjas, con cuál técnica se obtienen mejores resultados y justifique.**

**1. First Fit (Primera Ajuste)**

Recorre la lista de bloques libres desde el inicio y asigna el primer bloque que sea lo suficientemente grande para el proceso.

Ventajas:

* Rapido, porque no necesita recorrer todos los bloques.
* Simple en su implementación

Desventajas

* Puede dehar pequeños fragmentos de memoria al inicio de la lista. (fragmentación externa)

Información que debe mantener el Kernel.

Lista de bloques libres, usualmente en orden de dirección de memoria.

2. Best Fit (Mejor Ajuste)

Busca entre todos los bloques libres el que tenga el tamaño mas cercalo al requerido, minimizando el espacio desperdiciado.

Ventajas:

* Reduce la fragementacion interna, porque deja menos espacio sobrante.

Desventajas:

* Mas lento que el first fit, porque debe recorrer toda la lista de bloqyes
* Puede producir bloques pqueños que son inútiles (fragmentación externa)

Informacion que debe mantener el Kernel

Lista completa de bloques libres y sus tamaños-

**3. Worst Fit (Peor Ajuste)**

Asigna el bloque mas grande disponible. La idea es dejar bloques grandes disponibles para futuros procesos.

Ventajas:

* Puede evitar que queden demasiados bloques pequeños

Desventajas:

* En la práctica suele desperdiciar mucho espacio, creando fragmentación interna.

Informacion que debe mantener el Kernel

* Lista completa de bloques libres y sus tamaños-

**4. Next Fit (Siguiente Ajuste)**

Es una variación del First fit, pero no comienza desde el inicio cada vez. Comienza desde el ultimo bloque asignado y recorreo la lista hasta encontrar un bloque adecuado.

Ventajas:

* Distribuye la asignación de manera mas uniforme, evitando la congestión al inicio de la memoria.
* Algo mas rápido que el Best fit.

Desventajas:

* Puede recorrer gran parte de la lista antes de encontrar un bloque.

Informacion que debe mantener el Kernel

* Lista completa de bloques libres y un puntero al ultimo bloque asignado.

**Comparación según tipo de particiones**

| **Tipo de partición** | **Técnica más adecuada** | **Justificación** |
| --- | --- | --- |
| **Fijas** (tamaños predefinidos) | **First Fit o Next Fit** | La memoria está dividida en bloques de tamaño fijo; no hay mucho que optimizar, y First Fit es rápido y simple. |
| **Dinámicas** (tamaño variable) | **Best Fit** | Permite asignar bloques del tamaño más exacto, minimizando desperdicio y fragmentación interna. First Fit también se usa por velocidad, pero Best Fit suele usar menos memoria sobrante. |

💡 **Resumen clave:**

* **Kernel** necesita mantener **lista de bloques libres y sus tamaños**, y en algunos casos un puntero (Next Fit).
* **Particiones fijas:** simplicidad y velocidad → First Fit / Next Fit.
* **Particiones dinámicas:** eficiencia de espacio → Best Fit.

**29. Segmentación**

**a. Explique cómo trabaja este método de asignación de memoria.**

**b. ¿Qué estructuras son necesarias en el Kernel para poder ejecutarse sobre un Hardware que utiliza segmentación?**

**c. Explique, utilizando gráﬁcos, cómo son resueltas las direcciones lógicas a físicas.**

**d. En este esquema: ¿se puede producir fragmentación (interna y/o externa)?**

**e. De las técnicas enumeradas en el ejercicio 29 ¿Cuál se ajusta mejor para la asignación de memoria principal?**

La segementacion es un esquema de gestión de memoria donde cada proceso se divide en segmentos lógicos según su función o tipo de datos.

* **Código** (segmento de instrucciones)
* **Datos** (variables, estructuras)
* **Pila/Stack** (para llamadas y retorno de funciones)
* Otros segmentos específicos del programa.

Funcionamiento:

Cada segmento tiene un tamaño variable. La memoria se asigna a nivel de segmentos y no de procesos completos ni bloques fijos. Cada acceso a memoria tiene una dirección Logica (segmento + offset) que el Kernel traduce a dirección física.

Ventaja: los segmentos reflejan la estructura lógica del programa y permiten protección y compartición más sencilla.

**b. Estructuras necesarias en el Kernel**

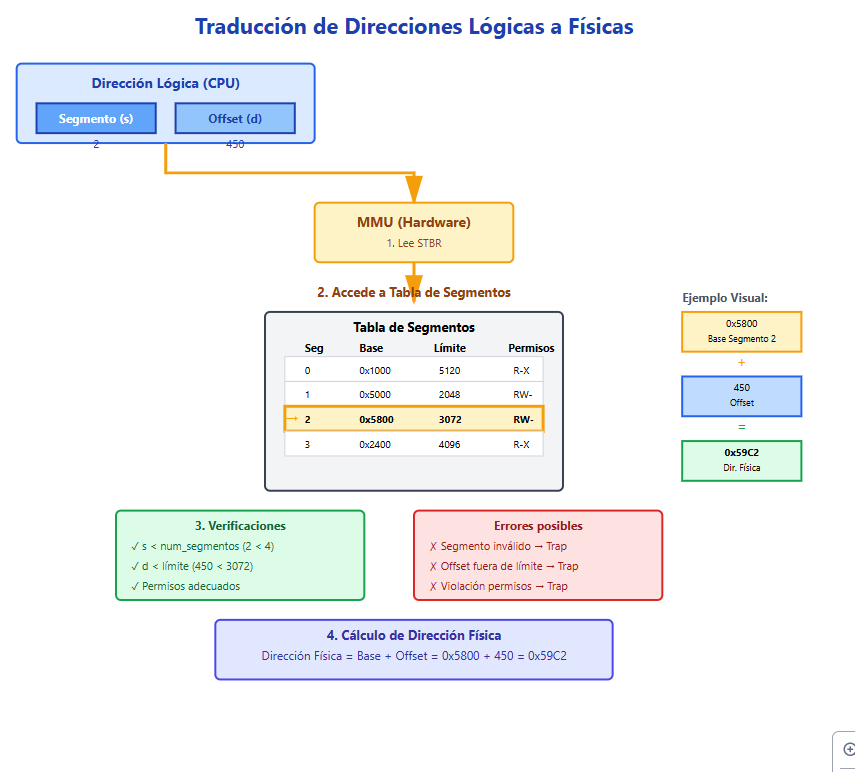
Para implementar la segmentación el Kernel debe mantener:

1) tabla de segmentos por proceso. Para cada segmento (dirección inicial en memoria y tamaño del segmento.)

2) registro de segmentación en el CPU. Contiene la base de los segmentos activos y permiten traducir direcciones lógicas a física.

3) lista de bloques libres en memoria principal (para ubicar segmentos que se cargan y liberan dinámicamente)

**c) Resolución de Direcciones Lógicas a Físicas**



**Proceso paso a paso:**

1. **CPU genera dirección lógica:** <s, d> → <2, 450>
   * s = número de segmento (2)
   * d = desplazamiento (450 bytes)
2. **MMU lee STBR** (Segment Table Base Register)
   * Obtiene dirección de la tabla de segmentos del proceso actual
3. **Accede a la entrada del segmento:**
   * Va a: tabla\_segmentos[s] → tabla\_segmentos[2]
4. **Verificaciones:**

✓ s < número\_de\_segmentos (2 < 4)

✓ d < límite (450 < 3072)

✓ Permisos adecuados

1. **Cálculo:**

Dirección Física = base + d

Dirección Física = 0x5800 + 450

Dirección Física = 0x59C2

1. **Acceso a memoria física** en dirección 0x59C2

**d. Fragmentación**

* **Fragmentación interna:** poco común en segmentación, ya que cada segmento tiene tamaño exacto según sus necesidades.
* **Fragmentación externa:** Sí, puede ocurrir, porque la memoria se asigna en bloques de tamaño variable y cuando los segmentos se cargan y liberan se crean huecos dispersos. Esto se puede mitigar con **compactación** (mover segmentos en memoria para juntar huecos).

**e. Técnica de asignación más adecuada**

* Para segmentación, **Best Fit** o **First Fit** son las técnicas más usadas:
  + **Best Fit:** minimiza los huecos de memoria (fragmentación externa).
  + **First Fit:** más rápido, sencillo de implementar.

Como la segmentación utiliza **bloques de tamaño variable**, **Best Fit** suele dar mejor aprovechamiento de memoria.